APPENDIX A

DESCRIPTION OF IPACK FORMAT FILE

The IPack Format File

The IPack format file is a text file that is created by a Windows utility program called IPackGen, that executes on your (the LAN administrator's) workstation after an installation package is created. The IPack file's input files are created by a program called Snapshot before the software is installed locally (during the creation of an installation package). These input files contain the following:

- Complete disk directory structure and contents (with file names, date-and-time stamps, file size, and so on)
- Saved copies of CONFIG.SYS, and all *.INI and *.BAT files.
- All Windows Shell (like Program Manager) group names and complete contents.
- Complete Windows Registration Database information.

To build the IPack format file, the IPackGen program compares the input files gathered by the Snapshot program with the similar information gathered by IPackGen after the software is installed. Using the differences, IPackGen creates the IPack format file.

LANInstall may also create a UPack, or Uninstall file for removing installations that have been made previously by LANInstall.

General Format

The IPack Format file consists of 11 groups, presented in no particular order. Each group is denoted by a name enclosed in double square brackets. Some of these groups may be empty and are listed below in alphabetical order:

[[AddedFiles]]

[[AddedRegDB]]

[[BasePersonality]]

[[Configuration]]

[[DeletedFiles]]

[[DeletedRegDB]]

[[MergedPersonality]]

[[ModifiedTextFiles]]

[[ModifiedWinShell]]

[[OtherErrorsAndWarnings]]

[[ReplacedFiles]]

note icon *Although the names are given in mixed case, both case and leading whitespace are ignored. (For example," [[ADDEDFiles]]" is acceptable.) However,all characters inside the brackets are significant, and must match exactly.*

Any line whose first character is a semi-colon (";") is considered a comment and is ignored. Commented lines are copied (for example, from IPack to Log, from Personality to IPack, from Log to UPack), with the following exceptions:

- All comments are removed from **[[OtherErrorsAndWarnings]]**, **[[AddedFiles]]**, **[[DeletedFiles]]**, and **[[ReplacedFiles]]** groups.
- Comments in **[[AddedRegDB]]** are copied into **[[DeletedRegDB]]** (and vice-versa) when UPGen creates an uninstall IPack Format file (for instance, a FilterUPack or LogUPack file).

- Comments in the **[[MergedPersonality]]** group (which is created by merging the Base Personality with a workstation Personality) will contain only those comments from the file that is being overwritten (for instance, acting as "base").. That is:
  - Comments in the **[InstallFlags]**, **[UninstallFlags]**, or **[FileTransferFlags]** section will be copied only from the Base Personality (since the workstation Personality "Flags" sections always overwrites the Base Personality "Flags" sections).
  - Comments in the **[PathMacros]** section will be copied only from whichever Personality's **[PathMacros]** section is overwritten (as determined by the **BaseMacrosOverwrite** flag).

IPack Format File Groups

The next several sections explain the groups that make up the IPack format file.

[[AddedFiles]], [[DeletedFiles]], [[ReplacedFiles]]

The**[[AddedFiles]]**, **[[DeletedFiles]]**, and **[[ReplacedFiles]]** groups contain lists of directories and filenames. The format consists of a full directory path, followed by filenames within that directory. Sometimes a group may be empty. Additional sets of directories and filenames can be given as needed.

Following is a sample **[[AddedFiles]]** group:

```
[[AddedFiles]]
$(WINDOWS)\W4W
    DECOMP.EXE
    SETUP.EXE
        .
        .
        .
$(WINDOWS)\W4W\CLIPART
    BOOKS.WMF
```

```
        CAPITOL.WMF
            .
            .
            .
$(WINDOWS)
    WINWORD.INI
```

Entries in the **[[ReplacedFiles]]** group are only valid within a Log file (see Appendix C for further discussion). Entries in this group are ignored during an installation. The IPackGen utility always places added and replaced files in the **[[AddedFiles]]** group. The Administrator's Log file tells whether a file was added or replaced during the initial installation that is part of creating an installation package.

IPackGen creates these groups in the IPack Format and Log files by comparing the data on all files and directories that are scanned before and after the initial installation. Any file or directory that changed (added, deleted, or replaced) is included in the appropriate group. The only exceptions to this are:

- The temporary working directory created by Snapshot (which is removed by IPackGen) is never included, nor are any files within that directory.
- No files that have both **hidden** and **system** attributes are included.
- Windows *.GRP files are not included (except for .GRP files that get added without a corresponding Windows Shell group).

During the installation, a check of the time and date stamps ensures that an existing file is replaced only if the added file is newer than the existing file. If it is not, an entry in the log file notes that the file was *not* replaced (unless the two files are identical). A Personality file flag, **OverwriteNewerFiles**, can be set to True to disable the time and date check, which forces the added file to be copied.

The **[[DeletedFiles]]** group is always processed first during an installation, followed by the **[[AddedFiles]]** group. Although unlikely, if a file appears in both lists, the existing file is deleted (if there is one) and the new file is added.

[[AddedFiles]]

During an installation, if a directory listed in the `[[AddedFiles]]` group does not exist, it is created.

In one type of IPack Format file, a FilterUPack, a "*" may appear after a directory name. A FilterUpack file is sometimes part of the deinstallation process. for more information, see "Uninstall IPack Files" in appendix D. The trailing `\*` indicates that all files listed under this directory in the LogUPack file being filtered are acceptable and may be added.

[[DeletedFiles]]

In the `[[DeletedFiles]]` group, a directory name followed by a list of filenames indicates that the listed files are to be deleted and that the directory (and any unnamed files) will remain. If no filenames are specified, no action will occur.

A directory name ending with `\*` indicates that the directory and all of its files will be deleted. (This includes all sub-directories and *their* files.) Any filenames following the directory name are ignored.

A directory name ending with `\?` indicates that the directory will be deleted only if it is empty after deleting all the other files and sub-directories listed in the **[[DeletedFiles]]** group.

[[AddedRegDB]] and [[DeletedRegDB]]

The **[[AddedRegDB]]** and **[[DeletedRegDB]]** groups are non-empty only for installations of Windows programs that modify the Registration Database. If one or both of these groups contain entries, the **Windows** macro must be defined (see Appendix B for details).

[[AddedRegDB]]

Entries in the **[[AddedRegDB]]** group have the following form:

key=value

key refers to the name of the key as stored in the Registration Database. It is similar to a directory path, but does not begin with the root indicator (\).

value is the string to the right of the equal sign, and may consist of any characters.

For example:

```
regedit\shell\open\command=regedit.exe %1
```

During an installation, keys listed in this group are added to the workstation's Registration Database if they do not already exist. If the key already exists, but has a different value than the value specified, the value is changed to the specified value. However, an existing value is *not* be replaced with a null value. (A null value is given by an entry of the form "`key=`", that is, no value is specified to the right of the equal sign.)

[[DeletedRegDB]]

Entries in the **[[DeletedRegDB]]** group have the following form:

key

Only *key* names are listed, since the value is not considered when deleting a key. (The Log file *does* contain the value, however, as described in Appendix C).

The keys given in this group are deleted, as are sub-keys branching from each specified key. For example, assume the Registration Database at a particular workstation contains the following keys:

```
key0
key0\key1
key0\key1\key2
key0\key1\key2\key3
key0\key1\keyTwo
key1
```

If `key0\key1` is listed in the **[[DeleteRegDB]]** group, the following keys are deleted:

```
key0\key1
key0\key1\key2
key0\key1\key2\key3
key0\key1\keyTwo
```

[[BasePersonality]]

The **[[BasePersonality]]** group usually contains a copy of the default Base Personality file present on the LAN admin's workstation during the creation of the IPack format file.

note icon *If the personality file specified in Parmfile.ss is not the default personality file on the administrator's workstation, the file copied is the one specified in the file Parmfile.ss.*

The **[PathMacros]** section is copied as read. Rather than copying all of the flags sections in the personality file, only the appropriate flags sections get copied (For a file transfer, this would be the **[FileTransferFlags]** section. The **[UninstallFlags]** section is copied also.)

During installation, this information is used only if a search of the workstation's Personality file fails to find a required path or flag entry. For more information on these entries, refer to Appendix B, *Personality Files*.

The only time the **[[BasePersonality]]** group contains information other than that read from the base Personality file is during the deinstallation process, within a UPack, LogUPack, FilterUPack, or a Log file. In these cases, the group's information still represents the base (default) Personality. However, the information is derived from sources other than the base Personality file. This is necessary to properly "undo" an installation where the Personality file has changed since the installation was performed.

[[Configuration]]

Thie **[[Configuration]]** group is always required to be non-empty and contains entries that pertain to the configuration of the package being installed. These entries are explained as follows:

ExcludePath=*Path*

The `ExcludePath=` entry specifies a path to a directory that contains files to be excluded from the installation. It must be created by hand because it is never created by the IPackGen utility. If the `ExcludeFiles` flag in a workstation's Personality file is set to True, no files from the specified directory are copied, nor is the directory created. (This also includes creation or modification of text files given in the `[[ModifiedTextFiles]]` group.) All directories that branch from the specified directory are also skipped.

The `ExcludePath=` entry may be given more than once if it is necessary to specify more than one directory path.

IPackDataFile=*Path\Filename*

The `IPackDataFile=` entry specifies the complete path and filename of the compressed "archival" file that contains copies of all files that must be copied during installation of the software package. This entry is created by IPackGen.

For more information, see Appendix C, *Log Files*.

IPackFileType=*Type*

The `IPackFileType=` entry identifies the type of the IPack Format file and is created by IPackGen (or DPackGen or UPGen, as appropriate). The `IPackFileType=` enty must contain one of the following values:

`IPack` An an installation IPack Format file.

`DPack` A file transfer (distribution) IPack Format file.

`UPack` A "merged" de-installation IPack Format file, created by merging a FilterUPack and LogUPack file.

`FilterUPack`
A file used to filter entries in a LogUPack file during the deinstallation process. "Filter" means to modify an event or prevent it from happening. FilterUPack is created from an IPack file during the deinstallation process.

`LogUPack` A UPack Format file created from a Log file during the deinstallation process.

`Log` A log file (created during installation, deinstallation, or file transfer).

`Error` One or more errors were encountered during generation of this file.

`ErrorFatal`
A fatal error was encountered, forcing processing to terminate prior to completion.

RequirementsFile=*Path\Filename*

The `RequirementsFile=` entry specifies the complete path and filename of the file that contains the hardware and software requirements for the software package being installed. IPackGen creates the `RequirementsFile=` entry by copying the corresponding entry given in the Parmfile.li file.

[[MergedPersonality]]

The **[[MergedPersonality]]** group contains a copy of the Personality file that was used during creation of this IPack Format file. It contains the macros and flags that resulted from the merging of the base Personality and the LAN Administrator's Personality.

This information is used by DPackZip, and by UPackGen when processing an IPack Format file directly (not a Log file). For more information, refer to Appenidx B, *Personality Files*.

[[ModifiedTextFiles]]

The **[[ModifiedTextFiles]]** group uses a script language to detail the changes that must be made to specific text files during the installation process. Text files that may be modified include *.BAT, CONFIG.SYS, *.INI, and possibly other files as determined by the creator of the IPack Format file (such files will be given a file type of INI).

note icon *Changes made to the "Order=" entry in the `[Settings]` section of the"PROGMAN.INI" file , as well as to the entire `[Groups]` section, should never be given in this group. Such changes must be made by using the appropriate commands available in the `[[ModifiedWinShell]]` group detailed later in this appendix.*

The **[[ModifiedTextFiles]]** group is created by IPackGen when it encounters a text file that has been added or changed. A deleted text file, however, is not entered in this group (instead, it appears in the **[[DeletedFiles]]** group). This does not prevent the deletion of a text file from occurring in this group. For example, a text file that was created during a LANInstall installation is "uninstalled" by a series of **Remove** commands. If the text file is empty after processing all the commands, it is deleted.

Notation Conventions and Rules

The following notation conventions pertain to the commands that may be included in the **[[ModifiedTextFiles]]** group:

- Text in *italics* denotes a replaceable parameter. It must be replaced with suitable information (see below). ***Bold*** text is used to denote keywords (like commands).
- Replaceable parameters shown in braces (like "{*Where*}") are optional.
- The OR character (|) between two elements indicates "one or the other, but not both."
- All text is case insensitive.
- Leading whitespace is ignored in most instances. However, whitespace is necessary to delimit tokens, except where a distinct delimiter exists (like a bracket ([]).
- A line may contain up to 510 characters. A command cannot continue on to the next line. A line whose first non-whitespace character is a semicolon is considered a comment and is ignored.
- If a command matches more than one line within a file (or within a section, if dealing with sections), the first line matched will be the line acted upon.

Replaceable Parameters

FindText

Locates a line in the original file by combining 1 or more {*Text*} search strings. For example, ({rem} {device} {mouse.sys}) will locate a line containing all three strings (in the order given).

FindText must be surrounded by parentheses. Consider the following example:

```
Append ({path}{=}) {$(WINDOWS)\ACCESS}
```

The search for a matching line containing *FindText* always begins at the top of the original file (or section, if a section is being modified). *FindText* locates the first matching line in the original file that remains unmodified in the new (working) file. In other words, if a line containing *FindText* is found, but that line has been **Delete**'d, **Rem**'d, or **Unrem**'d in the working file, the line will not be used, and the a search for the next matching line begins.

A line will match *FindText* only if the *FindText* strings occur in the line in the same order as given in *FindText*. For example, {key}{=} requires that the equal sign character occurs after the matched "key"string.

FindText matches are performed in the same manner as a search for a line to be added (see ***Add matches*** later in this chapter). Enough *FindText* must be given for at least a partial match; more information can be given to ensure an exact match. For example, given an AUTOEXEC.BAT file containing:

```
set var1 = i:
```

the *FindText*:

```
({set}{var1})
```

will match with the AUTOEXEC.BAT line. A more specific *FindText*, such as:

```
({set}{var1}{C:})
```

will not match this line, since `C:` represents the value portion of the line, and the existing line has a value of `j:`, not `C:`.

In order to match comment lines, the proper comment character(s) must be given as the first *FindText* item; comment lines will not match *FindText* that does not include the comment `path = c:\dos;c\windows`

The *FindText* (`{path}{=}`) only matches the second line, not the first.

Path/Filename

A full path, including drive letter and colon, followed by a filename (including extension, if any). May include one or more path macros (for example `$(DOSDRIVE)\FILE.EXT`).

Section

For files containing sections (like .INI files), `[Section]` must be specified in the first command following a **File** command; it is optional in subsequent commands applying to the same section. If the section doesn't exist in the original file, all commands except **Add** and **Append** will log an error and skip the command. **Add** and **Append** will create a new section if `[Section]` does not exist.

If the [*Section*] specified is commented out, the comment character(s) must appear as the first character of [*Section*] (for example, ;[fonts]).

Comment lines at the beginning of an .INI file, before any [*Section*], are referenced by using the special "null section" notation: "[]" (no characters between the open and close brackets).

The search for a matching section always begins at the top of the original file. Except for **Add** and **Append**, the matching section must also exist in the current file (the one being modified). If the section is in the original file, only a **Delete** command will remove it from the current file (or **Remove**, if the **RemoveIsDelete** flag is True).

Text

A complete or partial statement within a text file (like SYSTEM.INI, AUTOEXEC.BAT, and so on). It may include replaceable parameters, like $(DOSDRIVE), anywhere within the string.

Text must always be enclosed within braces. For example,

```
{This is a Text string.}
```

If *Text* contains braces, they must properly matched. The following is invalid:

```
{This is an open-brace: "{" but there is
no matching end-brace.}
```

Where

Where identifies the position in the text file the specified line is to be added. If *Where* is not given, the line will be added after the previously processed line. If no line has been processed yet in the file being modified, the line will be added to the end of the file. *Where* is not applicable to Windows .INI files, since they are not order-dependent. Additions made to these files are always made at the end of the current section (and sections will always be added to the end of the file being modified).

Where is replaced by one of the following commands:

After *FindText*	The line below the first line matching *FindText*.
Before *FindText*	The line above the first line matching *FindText*.

End	The last line of the file.
Next	The line after the previous line just processed (default if *Where* is not specified).
Start	The first line of the file.

As currently implemented, Next's "line just processed" includes a line processed by *any* command, not just **Add**. This means that if lines are being added to the end of a file, and then a line is **Rem**'d, the next line will be added after the **Rem**'d line (unless a *Where* clause is given).

$(*MacroName*)
A macro name to be replaced with a path during the actual installation (based on the macro's value in the Personality file on the workstation where the installation is taking place).

[[ModifiedTextFiles]] Commands

This section details the commands used in the **[[ModifiedTextFiles]]** group.

File *Path\Filename*

The **File** command denotes that any following commands are to be applied to the specified file. This command is required, and must appear as the first statement for each file to be modified. *Filename* must be CONFIG.SYS, *.BAT, or *.INI. Any other filename will be processed as if it is an .INI file, which may lead to errors.

The following commands appear after a **File** command, and apply to the specified file:

Add {[*Section*]} {*Where*} *Text*
This command adds the new line *Text* to the current file being modified. **Add** must be replaced by one of the following:

Add!	Adds the new line even if an identical line already exists in the current file.
Add+	Adds the new line as long as an identical line does not already exist.

Add Adds the new line if no similar line exists, otherwise the existing line will be **Remove**'d ("translating" to either **Delete** or **Remout**) and the new line added.

Add? Adds the new line only if no similar line exists.

Three of the four forms of the **Add** command add a new line without disturbing existing lines. The fourth form, "**Add** " may add a new line, or replace an existing line. When searching for an existing similar line, the entire original file is searched (before changes). Just as with *FindText* searches, only matching lines that are unmodified in the new (working) file are used. If a *Where* clause designating a specific location is given (**before** or **after** a specific line), then only that specific line will be examined for a match with the line being added—the entire file will not be searched. (This ensures that the line is added where specified, regardless of whether the line is new, or it replaces an existing line.) If one of the other *Where* clause locations is given (**end**, **next**, or **start**), the entire file is searched. In this case, the *Where* location will be used only if the line is new.

For a description of matching "identical" and "similar" line, see "**Add** matches" later in this chapter .

Add creates a new section if *Section* does not already exist. An empty section is be created if only "**Add** [*Section*]" is given (and *Text* is not given). (However, even the command "**Add!** [section]" will not add [*Section*] if it already exists.)

This command has the opposite effect of the **Delete** command (that is, a **Delete** command in an Install Log file will be translated to a corresponding **Add** command for inclusion in a UPack format file).

Append {[*Section*]} *FindText Text*

Appends *Text* to the end of the first statement found that matches *FindText* within the current file (or section). The first character of *Text* is assumed to be the delimiter between items being appended. The installation routine ensures that two consecutive delimiters are not placed together. For example, consider the following command line:

```
Append ({path}) {;c:\windows;d:\windows\winword}
```

If the file being modified has a path statement ending with a semicolon, the first semicolon in the **Append** command line will not be added.

Multiple items may be appended by including them within *Text*, separated by the `append` delimiter. If an item already exists in the line matching *FindText*, it will not be appended. (The comparison for an existing item is case insensitive.) The following example appends only `calendar.exe` to the line `load=clock.exe calc.exe cardfile.exe`:

```
Append ({load}) {calendar.exe CALC.EXE Cardfile.exe}
```

If the **AppendAndExtractSaveOriginal** flag in the Personality file is set to True, a commented-out copy of the original (unmodified) line is **Add**'d to the file being modified.

If no line matching *FindText* is found, a line created from the **Append** command line is added to the end of the file (or section). The line is created by concatenating the *FindText* elements, each separated by a space, followed by an equal sign character, and *Text* (without the first [delimiter] character). Consider the following example:

```
Append ({set}{temp}) {;$(WINDEV)}
```

It creates the following line if no line containing "set" and "temp" is found:

```
set temp=$(WINDEV)
```

Although the line was added, it appears in the Log file as an **Append** operation.

This command is used as the opposite of the **Extract** command (that is, an **Extract** command in an Install Log file will be translated to a corresponding **Append** command for inclusion in a UPack format file).

Delete {[*Section*]{? | *}} *FindText*

The **Delete** command deletes the first line encountered containing *FindText* within the file (or section). The original file's contents will be searched (as opposed to the current file after modifications).

If [*Section*] is not given but the file being modified contains sections, the last *Section* specified will be searched. (If a *Section* has not yet been specified for this file, no delete will occur.)

A '*' given after [*Section*] name (without *FindText*) specifies that the entire section be deleted, including all entries within that section.

A '?' given after [*Section*] specifies that the section be deleted only if it is empty (i.e., contains no entries, including comments, but not blank lines). If it is followed by *FindText*, then the specified line will be deleted before the check for an empty section is made.

Delete [*Section*] (without a '?', '*', or *FindText*) is invalid and will be ignored.

Extract {[*Section*]} *FindText Text*

Removes *Text* from the first statement found matching *FindText* within the current file (or section). The **Extract** operation fails if a statement matching *FindText* is not found.

The first character of *Text* must be the "append delimiter" between items being extracted. Multiple items may be extracted by including them within *Text*, separated by the "append delimiter." The comparison for an existing item is case insensitive. Consider the following example:

```
Extract ({load}) { calendar.exe CALC.EXE Cardfile.exe}
```

It extracts `cardfile.exe` and `calc.exe` from the following line:

```
load=clock.exe calc.exe cardfile.exe
```

If the **AppendAndExtractSaveOriginal** flag in the Personality file is set to True, a commented-out copy of the original (unmodified) line will be **Add**'d to the file being modified.

The **Extract** command is used as the opposite of the **Append** command (that is, an **Append** command in an Install Log file will be translated to a corresponding **Extract** command for inclusion in a UPack format file). The **Extract** command allows text that was previously added using **Append** to be removed from within the same line, no matter where it currently is within that line.

Remout {[*Section*]{?|*}} *FindText*

Creates a remark from the first line encountered containing *FindText* within the file (or section). (If the line is already commented out, no action will occur.) Remarks are preceded by "rem" unless the file is an .INI file, in which case a semicolon appears.

If [*Section*] is not given, but the file being modified contains sections, the last *Section* specified is searched. (If a *Section* has not yet been specified for this file, no action will occur.)

A '*' given after [*Section*] name (without *FindText*) specifies that the entire section be commented out, including all entries within that section.

A '?' given after [*Section*] specifies that the section heading (name) be commented out, if the section is empty (commented and blank lines are not considered in determining whether the section is empty or not).

Remout [*Section*] (without a '?', '*', or *FindText*) comments out the section heading only.

Remout is used as the opposite of the **Unrem** command.

Remove {[*Section*]{? | *}} *FindText*

The **Remove** command removes the specified line by acting as either a **Delete** or **Remout** command. By default, **Remove** acts as a **Remout** command, commenting out the first line encountered containing *FindText* within the file (or section). If the line is already commented out, no action will occurs. The original file's contents will be searched, as opposed to the current file after modifications. However, if the **RemoveIsDelete** flag in the Personality file is set to True, then **Remove** will actually **Delete** the specified line.

If [*Section*] is not given but the file being modified contains sections, the last *Section* specified is searched for a *FindText* match. If a *Section* has not yet been specified for this file, no remove occurs.

A '*' given after [*Section*] name (without *FindText*) specifies that the entire section will be removed, including all entries within that section.

A '?' given after [*Section*] specifies that the section be removed only if it is empty (contains no non-blank entries, or comments if the **RemoveIsDelete** flag is True). If it is followed by *FindText*, then the specified line will be removed before the check for an empty section is made.

Remove never appears in the Log file; it is always changed to **Delete** or **Remout**, depending on which action really occurred. Defining **Remove** this way allows the LAN Administrator to select a default action for **Remove** and yet manually override specific instances by editing the IPack Format file. For example, if the **RemoveIsDelete** flag is set to False, but you know that the command **Remove** {Files=40} can safely delete the line "Files=40", you can change the command to **Delete**.)

Remove is used as the opposite of the **Add** command (that is, an **Add** command in an Install Log file will be translated to a corresponding **Remove** command for inclusion in a UPack format file). Since **Remove** itself never appears in a Log file, it has no opposite command.

Unrem {[*Section*]}{*} *FindText*

Removes a remark from the first line encountered containing *FindText* within the file (or section). The remark characters removed will be "rem" unless the file is an .INI file, in which case ";" will be removed. As mentioned in the *FindText* discussion in "Replaceable Parameters" earlier in this appendix, the remark character(s) must be given as the first *FindText* item in order to find the line to **Unrem**.

If the specified [*Section*] refers to a section heading (name) that is commented out in the original file, the comment character(s) must be given as the first character (for example, in an .INI file `;[Colors]`).

If [*Section*] is not given but the file being modified contains sections, the last *Section* specified will be searched. If a *Section* has not yet been specified for this file, no action occurs.

A '*' given after [*Section*] name (without *FindText*) specifies that the entire section be uncommented, including all entries within that section.

Unrem [*Section*] (without a '*' or *FindText*), uncomments the section heading only.

Unrem is used as the opposite of the **Remout** command

[[ModifiedTextFiles]] Examples

Consider the following **[[ModifiedTextFiles]]** section of an IPack format file:

```
[[ModifiedTextFiles]]
FILE $(DOS)\AUTOEXEC.BAT
APPEND ({path}{=}) {$(WINDOWS)\ACCESS}
ADD NEXT {SET INIT=$(WINDOWS)\ACCESS}
ADD NEXT {SET TEMP=$(WINDOWS)\ACCESS\TEMP}
FILE $(WINDOWS)\SYSTEM.INI
ADD [386Enh] {device=*vmcpd}
REMOVE ({device}{=}{vmcpd})
```

The preceding commands would make the following changes to the AUTOEXEC.BAT file found in the `$(DOS)` directory.:

- The **Append** command adds the base path to the end of the current "path" variable found in AUTOEXEC.BAT. If the flag **AppendAndExtractSaveOriginal** in the Personality file is set to True, a commented-out copy of the original `path=` statement is added.
- The line `{SET INIT=$(WINDOWS)\ACCESS}` is added after the `path=` statement. If a `set init=` statement already exists in this position within the file, it is **Remove**'d. If the flag **RemoveIsDelete** in the Personality file is set to False (or is not defined), then a **Remout** of the current `set init=` statement is performed, otherwise a **Delete** occurs.
- The line `SET TEMP= $(WINDOWS)\ACCESS\TEMP}` is added after the `set init=` statement. A **Remove** of an existing similar line occurs as just described.
- The line `device=*vmcpd` is added to the end of the `[386Enh]` section of SYSTEM.INI. If a line containing `device=` and `vmcpd` already exists in this section, it is **Remove**'d.
- The **Remove** command operates on the "current" section (`[386Enh}`). It affects the original `device=*vmcpd` statement, not the one added by the preceding command. If the flag **RemoveIsDelete** in the Personality file is set to True, the specified statement is deleted. If the **RemoveIsDelete** flag is set to False, a **Remout** of the statement occurs. This command is redundant because the existing similar line would have been **Remove**'d during the **Add** command described earlier.

Assume CONFIG.SYS is being modified, and it contains the following line:

```
DEVICE=$(WINDOWS)\HIMEM.SYS
```

The following statement:

```
ADD {device=$(DOS)\HIMEM.SYS}
```

Replaces the line:

```
DEVICE=$(WINDOWS)\HIMEM.SYS
```

with:

```
device=$(DOS)\HIMEM.SYS
```

after making the appropriate substitutions of `$(WINDOWS)` and `$(DOS)`. The Log file will show[1]:

```
Delete {DEVICE=$(WINDOWS)\HIMEM.SYS}
Add {device=$(DOS)\HIMEM.SYS}
```

This example assumes that the flag **RemoveIsDelete** in the Personality file is set to True (if not, the **Delete** would be a **Remout**). and illustrates that an **Add** command performs a **Remove** on an existing line (if there is one). The example also assumes that paths are used in determining a match (see **Add matches**, later in this appendix). If they are not, then no change is made, since the root program name in the line to be added is the same as that of the existing line.

Here is another example:

```
Add [fonts]
Add {Small Fonts (VGA res)=SMALLE.FON}
```

The sequence adds the following line to the end of the [fonts] section in the current file:

```
Small Fonts (VGA res)=SMALLE.FON
```

If a section named [fonts] does not already exist, it is created. "**Add** [fonts]" is written to the Log file as "**Add** [fonts] +" if the section was created.

Add matches

[1]The actual macro name used in the **Add** statement log may be different from that given in the Ipack Format, since it depends upon the workstation's Personality file.

The **Add** command searches the "original" copy of the current text file being modified for a line that matches the line being added, in order to determine what action to perform (replace the current matching line or just add the new line). It searches for a match in the original copy, and then references the matching line's corresponding line in the "working" copy.)

The range of lines that may be searched is limited as follows:

- If a *Where* clause designating a specific location is given (before or after a specific line), then only that specific line is examined for a match with the line being added
- If a *Where* clause is not given and the file contains sections, only the current section is searched.
- If a *Where* clause is not given and the file does not contain sections, all lines in the entire file are searched.

A match is either "identical" (also, "exact") or "similar" (also, "partial"). Whether a match is identical or similar is determined by the type of file being modified (CONFIG.SYS, *.BAT, or *.INI), as well as the specific line being added. Different line types have been defined, grouping together lines with similar characteristics. A line is composed of several elements; a specific line type has some or all of these elements. The exact meaning of the element may also depend upon the specific line type (or even the specific line).

The possible elements of a line are:

keyword
The keyword might be the command in a *.BAT file (for example, "ECHO"), or the key in an *.INI file (for example, "speed" in the line `speed=medium`).

key modifier
An additional piece of information required to make the keyword unique. Both the keyword and key modifier (when present) must be identical for a line to match. The key modifier is not applicable to *.INI files.

path
If a specified path is necessary to further qualify a line, it is stored in "path" (as opposed to being part of "value"). For example, the command `call path\progname` uses *path* as a unique part of the statement. *Path* is not applicable to *.INI files, which always use "value".

value

The remainder of the line, after the other elements have been identified. For example, "medium" in the *.INI line `speed=medium`, or `/e:1000 ./p` in the CONFIG.SYS command

```
shell=D:\msdos\command.com /e:1000 /p.
```

The different line types, the elements they include, and an example of each type are summarized in the following table.

Line Type Name	Keyword	Key Modifier	Path	Value	Example
Blank					Blank lines: always exact matches
BatStd		Partial	Exact	Exact	C:\Windows\smartdrv C 2048
CfgDevice	Exact	Partial	Exact	Exact	Device=C:\DOS\setver.exe C:
IniStd	Partial			Exact	speed=medium
KeyAndKeyMod	Partial	Partial		Exact	SET VAR1 = 2000
KeyAndProdName	Partial	Partial	Exact	Exact	Call D:\bats\dos_path DOS_MSC
KeyAndValue	Partial			Exact	Echo This is a test

Partial	Indicates that the element must match between the lines being compared in order for the lines to be considered at least "partial" matches.
Exact	Indicates that the element must match in order for the lines to be considered "exact" matches.

The following table shows how to determine the line type of a line based on the type of file the line is in, and the key word found in the line. It also lists the default **Add** command that IPackGen generates (for example, **Add!** is generated when adding a blank line).

File Type	Keyword	Line Type	Add Type
All files	<none>	Blank	!
CONFIG.SYS	Device	CfgDevice	
	Devicehigh	CfgDevice	
	Drivparm	KeyAndKeyMod	
	Install	KeyAndProgName	
	Shell	KeyAndProgName	

	<all others>	KeyAndValue	
*.BAT files	Break	KeyAndValue	
	Call	KeyAndProgName	
	Cls	KeyAndValue	!
	Command	BatStd	?
	Echo	KeyAndValue	!
	Emm386	BatStd	?
	Fastopen	BatStd	?
	Lastdrive	KeyAndValue	
	Goto	KeyAndValue	!
	If	KeyAndValue	!
	Loadhigh	KeyAndProgName	
	Lh	KeyAndProgName	
	Mirror	BatStd	?
	Path	KeyAndValue	
	Pause	KeyAndValue	!
	Set	KeyAndKeyMod	
	Share	BatStd	
	Shift	KeyAndValue	!
	Verify	KeyAndValue	
	<all others>	BatStd	+
*.BAT files	Device	KeyAndProgName	
	<all others>	IniStd	

The "@" character (as the first non-whitespace character of a *.BAT line) is not considered significant when searching for matching lines. For example, the line "echo Hello!" being added with the command:

```
Add{echo Hello!}
```

will match exactly with the following line in a *.BAT file:

```
@echo Hello!
```

Comment lines only match other, identical comment lines. For example, the line "files = 50" with the command:

```
Add{files = 50}
```

will not even partially match with the following line in a CONFIG.SYS file:

```
rem files=50
```

The lines "rem files = 50" and "rem files=40" do not even partially match.

[[ModifiedWinShell]]

The **[[ModifiedWinShell]]** group will be non-empty only for installations of Windows programs that modify the Shell's group information. This group is one of the last two groups to be processed, after all other files have been copied and modified as needed.

Notation Conventions and Rules

The following notation conventions pertain to the commands that may be included in the **[[ModifiedWinShell]]** group:

- Text in braces ("{~~~}") is optional.
- Text in *italics* is replaced with suitable information (see below). **Bold** is used to denote keywords (like commands).
- All text is case insensitive.
- Leading whitespace between tokens (keywords) is ignored.
- A Command can not continue on to the next line. A command may contain up to 510 characters. A line whose first non-whitespace character is ";" is considered a comment and is ignored.

[[ModifiedWinShell]] Commands

The **[[ModifiedWinShell]]** commands modify group files. Quotation marks must be used to delimit string arguments that contain spaces, commas, or parentheses. The syntax and command names were adapted from the Windows' Shell-DDE interface.

AddItem(CmdLine,Name{,IConPath{,IconIndex{,XPos{,YPos{,DefDir{,HotKey{,fMinimize}}}}}}})

Adds an item (i.e., icon) to the currently active group. If an item already exists with the same *Name*, the existing item will be replaced by this item.

CmdLine

A string that specifies the full command line required to start the application. It can be the full path of an executable file along with parameters required by the application. It can also be an associated file name. The association is taken from the registration database.

Name

A string that specifies the item title to be displayed below the icon in the group window.

IconPath

A string that identifies the filename for the icon to be displayed in the group window. It can be either a Windows executable file or an icon file. If this parameter is not specified, Windows will attempt to use the *CmdLine* parameter. If *CmdLine* specifies neither an executable file nor an associated executable file, Windows will use a default icon.

IconIndex

An integer that specifies the index of the icon in the file identified by the *IconPath* parameter.

XPos,YPos

The *xPos* and *yPos* integers specify the horizontal and vertical positions of the icon in the group window. If these values are given as "-1", or no arguments are specified after *IconIndex*, the icon will be added in the next available position in the group window.

DefDir

A string that specifies the name of the default (or working) directory.

HotKey

A decimal integer that identifies a hot (or shortcut) key sequence specified for the application.

fMinimize

A Boolean flag (0 or 1) that specifies whether an application window should be minimized when it is first displayed.

CreateGroup(*GroupName*)

Creates a new group, or activates the window of an existing group. If the group exists, this command is written to the Log file as **ShowGroup** instead of **CreateGroup**. (**CreateGroup** appears in the log file only if the group was actually created.)

GroupName

A string that identifies the group to be created or activated.

DeleteGroup(*GroupName*){?}

Deletes an existing group, including all the items (icons) within the group. However, if a '?' is given after the closing parenthesis, the group is deleted only if it is empty at the time the command is processed.

GroupName A string that identifies the group to be deleted.

DeleteItem(*ItemName*)

Deletes the first item in the currently active group found with the name of *ItemName*. If deletions are the only actions to be made from the currently active group, activate **DeleteItem** it is with **ShowGroup** rather than **CreateGroup** (see **ShowGroup** later in this appendix).

ItemName

A string that specifies the item to be deleted from the currently active group.

ReplaceItem(*ItemName*)

Deletes the first item in the currently active group found with the name of *ItemName*, and remembers the position of this item (icon). The next command should be an **AddItem** command, which will be placed in the same position as the item just deleted, regardless of the **AddItem**'s *XPos* and *YPos* positions (if any are given).

ItemName

A string that specifies the item to be replaced in the currently active group.

ShowGroup(*GroupName*)

The **ShowGroup** command activates the specified group window. Unlike **CreateGroup**, the specified window is not created if it does not exist. **ShowGroup** should be used in place of **CreateGroup** when only deletions are being made from a group (to prevent creation of an empty window).

[[ModifiedWinShell]] Example

Here is an example of the **[ModifiedWinShell]]** group that creates a new group "Windows Applications", and adds a new program item (i.e., icon) "Win App" to this group.

```
[[ModifiedWinShell]]
CreateGroup("Windows Applications")
AddItem(winapp.exe,"Win App",winapp.exe,2)
```

[[OtherErrorsAndWarnings]]

The **[[OtherErrorsAndWarnings]]** group contains error and warning messages that do not directly relate to any other group.

APPENDIX B

DESCRIPTION OF PERSONALITY FILE

The Personality File

A LANInstall Personality file is a text file containing workstation-specific information that allows an installation to be customized for a particular workstation. On any given workstation, this Personality file is optional. A base, or default, Personality file is included in the IPack format file. For more information on this default file, see the **[[BasePersonality]]** group description in Appendix A.

A Personality file contains path macros and customization flags that are described in this chapter. If a workstation's Personality file does not contain a particular macro or flag, the value in the IPack format file's **[[BasePersonality]]** group is used. If the same entry is in both the workstation's Personality file and the **[[BasePersonality]]** group, the value of the **BasePersonalityOverwrites** flag determines which entry is used. For normal operation, this flag should be set to FALSE in the **[InstallFlags]** and **[FileTransferFlags]** sections of the Personality file, and TRUE in the **[UninstallFlags]** section.

[PathMacros]

Specified path macros replace absolute paths, which allows you to customize installation parameters for individual workstations. They are defined in the Personality file section named **[PathMacros]**, and have the form `MacroName=path`. Consider the following example:

```
[PathMacros]
UTILS=F:\UTILS
DOS = d:\msdos

WINDRIVE=G:
Windows=$(WINDRIVE)\windows
BOOTDIR=C:\
```

Case is ignored, but non-leading whitespace is significant. The Macro name on the left side of the equal sign must contain the same whitespace as found in the macro reference. For example, `$(WindowsDir)` matches `WindowsDir`, but not `Windows Dir`).

Path macros are entirely user-defined and are not created by LANInstall. Macro names may be anything you like, but there must be a macro named **Windows** defined if the installation involves a Windows program. The **Windows** macro must expand to the complete path of the directory in which Windows is installed. For example, `Windows=$(WINDRIVE)\windows`.

After final macro expansion, directory paths must be *full*, that is, they must always begin with a root directory, usually preceded by a drive letter and colon. As an example, say the Personality file contains the following line:

```
WINUTIL=J:\WINDOWS\UTILS
```

Now say that the (absolute) path of the installed package is:

```
J:\WINDOWS\UTILS\PBS
```

The resulting "macro-based" path would be:

```
$(WINUTIL)\PBS
```

If the macro-based path is set to an absolute path (such as `J:\WINDOWS\UTIL\PBS`), the software is installed in the same, absolute location on all workstations and warning message is written to the Log file.

A valid path does not end with a "\", except when referring to the root directory (like BOOTDIR=C:\) Then it is absolutely essential. If such a macro is defined without the "\" character, it refers to the specified drive, not the drive's root directory.

When possible, you should avoid defining multiple macros for the same path. For example, assume the Base Personality has a path macro defined as `Util = C:\util`. A workstation's Personality has `Util` and another (say, `UtilPath`) set to the same path. After an installation, the workstation's installation Log file may show `UtilPath` as the macro used, rather than `Util`. This is not likely to cause problems, but it certainly could cause confusion.

[FileTransferFlags], [InstallFlags], [Uninstall Flags]

The **[FileTransferFlags]**, **[InstallFlags]**, and **[UninstallFlags]** sections of the Personality file contain Boolean flag values that can be included to personalize the procedure being performed (file transfer, installation, or uninstallation) for the workstation containing the Personality file. The following values are all acceptable (case is ignored):

True	False
T	F
Yes	No
Y	N
1	0

Default values are always False if the entry is not present, or a value is not given. Although the following flag names are shown in mixed case, case is not significant. Some flags may not be applicable to all sections. For example, **AppendAndExtractSaveOriginal** is ignored if it appears in the **[FileTransferFlags]** section.

AppendAndExtractSaveOriginal=*Bool*

If True, the **Append** and **Extract** commands of the **[[ModifiedTextFiles]]** group create (**Add**) a commented-out copy of the original line (before modification).

BackupOldVersion=*Bool*

If True, an archival copy containing all files that appear (uncommented) in the Log's **[[DeletedFiles]]** and **[[ReplacedFiles]]** groups is created, using the path and filename specified in the `iPackDataPath` parameter of the PARMFILE.LI file.

BaseMacrosOverwrite=*Bool*

If True, a macro in the `[[BasePersonality]]` group in the IPack format file takes precedence over the same macro in an individual workstation's Personality file. By default, if this entry is not present, the workstation's Personality file overwrites duplicate macros in the `[[BasePersonality]]` group.

For install and file transfer operations, this flag should normally be set to False. For an uninstall, this flag should normally be set to True. This ensures that the macros used are the macros that were in effect during the installation of the package that is being uninstalled. Set the **BaseMacrosOverwrite** flag to False for an uninstall when it is necessary to use the current definitions of the macros, rather than their old values.

DisableInstall=*Bool*

If True, this workstation refuses installations.

DisableModifiedTextFilesBAK=*Bool*

If this flag is False, and a file in the **[[ModifiedTextFiles]]** group is changed, the original file is renamed with a .BK extension. If this flag is set to True, the original file is overwritten by the changed file and no backup is made.

ExcludeFiles=*Bool*

If True, the directory or directories specified in all **`ExcludePath`** entries in the IPack format file's **[[Configuration]]** group are not created, nor are any files in those directories copied.

LogAllWarnings=*Bool*

If True, low-priority (nuisance) warnings are logged. By default, only high-priority warnings are logged (and all errors).

OverwriteNewerFiles=*Bool*

If True, files in the **[[AddedFiles]]** group are always copied, even if a file with a newer time/date stamp already exists on that workstation. If False, files is copied only if there is no newer existing file.

RemoveIsDelete=*Bool*

If True, the **Remove** command in the **[[ModifiedTextFiles]]** group of the IPack format file executes. If set to False, a **Remout** occurs. For more information, see **"[[ModifiedTextFiles]]"** in Appendix A.